

1 `cpp programa.c programa.i`

- a) ¿Qué ocurre cuando se invoca el comando `cpp` con esos argumentos?
Se genera un archivo llamado "programa.i".
- b) ¿Qué similitudes encuentra entre los archivos `programa.c` y `programa.i`?
Las similitudes que encuentro son que, en la última parte del archivo `programa.i` se encuentra el código de `programa.c` pero con los valores de las directivas sustituidos dentro del `main` y sin comentarios.
- c) ¿Qué pasa con las macros y los comentarios del código fuente original en `programa.i`?
Se sustituyen en los lugares correspondientes y los comentarios desaparecen.
- d) Compare el contenido de `programa.i` con el de `stdio.h` e indique de forma general las similitudes entre ambos archivos.
De manera general, ambos contienen definición de tipos y dentro de `programa.i` encontramos `linemarkers`, estos le indican al compilador que el código que sigue a continuación pertenece originalmente al archivo `stdio.h`, por lo que ambos archivos están enlazados.
- e) ¿A qué etapa corresponde este proceso?
Corresponde a la etapa de preprocesamiento.

2 `gcc -Wall -S programa.i`

- a) ¿Para qué sirve la opción `-Wall`?
Esta opción le indica al compilador que muestre todos los warnings (advertencias)
- b) ¿Qué le indica a `gcc` la opción `-S`?
Le indica al compilador que detenga el proceso después de la etapa de compilación pero antes del ensamblado.
- c) ¿Qué contiene el archivo de salida y cuál es su extensión?
Contiene el código ensamblador correspondiente a mi arquitectura (x86_64) y la extensión es `.s`
- d) ¿A qué etapa corresponde este comando?
A la etapa de compilación.

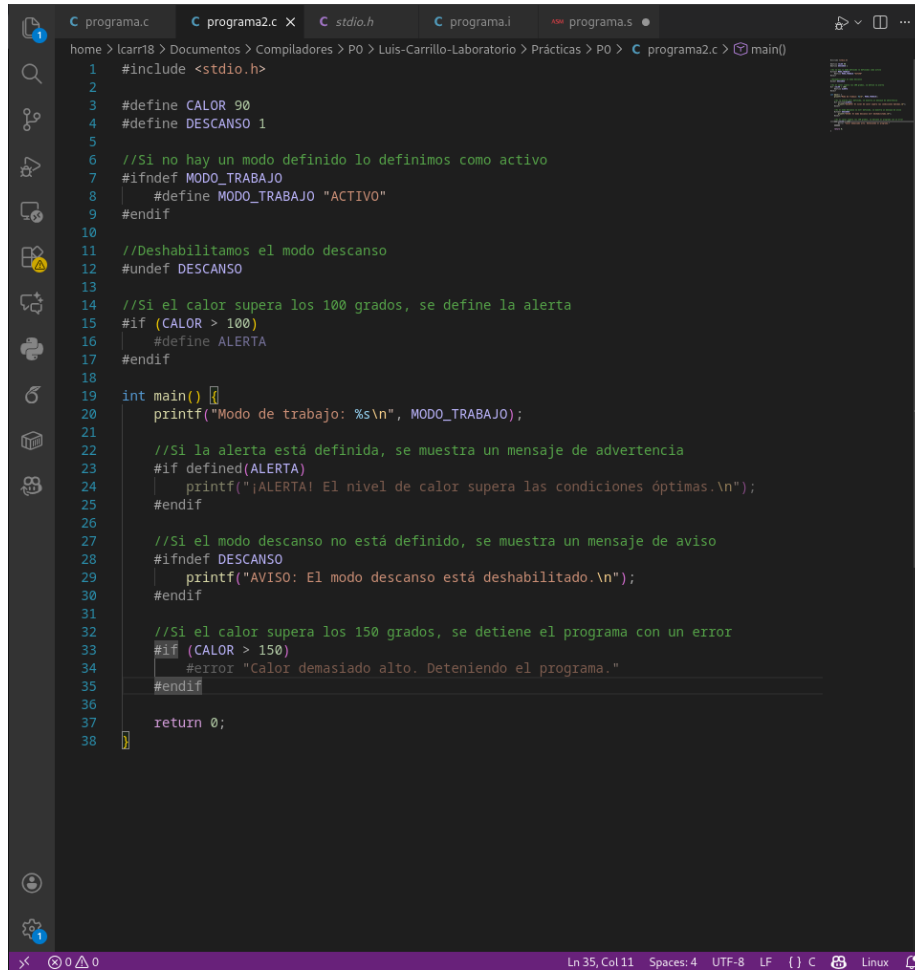
3 `as programa.s -o programa.o`

- a) Antes de revisarlo, indique cuál es su hipótesis sobre lo que debe contener el archivo con extensión `.o`.
como "as" corresponde a un comando relacionado con ensamblador, supongo que ahora `programa.o` contiene el resultado de que la máquina procese el código ensamblador, es decir, código binario.
- b) Diga de forma general qué contiene el archivo `programa.o` y por qué se visualiza de esa manera.
Dado que, efectivamente, el contenido de `programa.o` es código binario los editores de texto no pueden procesar esto.
- c) ¿Qué programa se invoca con `as`?
Se invoca al ensamblador del sistema operativo, en este caso el de GNU.
- d) ¿A qué etapa corresponde la llamada a este programa?
A la etapa de ensamblado.

4

Después de eliminar el comentario de la macro el resultado en la ejecución final no cambió, esto es debido al redondeo que se hace al imprimir el resultado, pues en realidad 3.1415926535897 y 3.1416 a nivel de operaciones con punto flotante se terminan redondeando a lo mismo.

5 Programa nuevo



```
1 #include <stdio.h>
2
3 #define CALOR 90
4 #define DESCANSO 1
5
6 //Si no hay un modo definido lo definimos como activo
7 #ifndef MODO_TRABAJO
8     #define MODO_TRABAJO "ACTIVO"
9 #endif
10
11 //Deshabilitamos el modo descanso
12 #undef DESCANSO
13
14 //Si el calor supera los 100 grados, se define la alerta
15 #if (CALOR > 100)
16     #define ALERTA
17 #endif
18
19 int main()
20 {
21     printf("Modo de trabajo: %s\n", MODO_TRABAJO);
22
23     //Si la alerta está definida, se muestra un mensaje de advertencia
24     #if defined(ALERTA)
25         printf("¡ALERTA! El nivel de calor supera las condiciones óptimas.\n");
26     #endif
27
28     //Si el modo descanso no está definido, se muestra un mensaje de aviso
29     #ifndef DESCANSO
30         printf("AVISO: El modo descanso está deshabilitado.\n");
31     #endif
32
33     //Si el calor supera los 150 grados, se detiene el programa con un error
34     #if (CALOR > 150)
35         #error "Calor demasiado alto. Deteniendo el programa."
36     #endif
37
38     return 0;
39 }
```

En general:

- `#ifndef`: Ejecuta el código si una macro no ha sido definida.
- `#undef`: Elimina la definición de una macro previamente definida.
- `#error`: Detiene la compilación inmediatamente y muestra un mensaje de error personalizado.
- `#if`: Ejecuta una sección de código si se cumple una condición.

En este programa:

- `#ifndef`: Asegura que el programa siempre tenga un `MODULO_TRABAJO`. Si no estaba definido se define como "ACTIVO", evitando que el programa falle por falta de datos.
- `#undef`: Simula una anulación manual de permisos.
- `#error`: Se detiene todo y se lanza un mensaje, impidiendo que se genere un ejecutable.
- `#if`: Si el `CALOR` es mayor a 150 se ejecuta error (explicado anteriormente) para que nadie pueda ejecutar con más de 150 de `CALOR`.